Experiment No.: 03

Aim: Create a Cryptocurrency using Python and perform mining in the Blockchain created.

Task to be performed :

1. Download the code from folder, **Lab_3**
2. Install requests in the virtual environment created in the Lab 2. (Follow the instructions)
3. Run the files - **hadcoin_node_5001.py**, **hadcoin_node_5002.py**, **hadcoin_node_5003.py** in 3 different terminals.
4. Open Postman, from each node - invoke **connect_node()** and pass the peers as POST requests.
5. Perform the following functions
   - Add Transactions - invoke add_transactions() as a POST request.
   - mining - mine_block(),
   - fetch the chain - get_chain(),
   - replace the longest chain - replace_chain()
6. Modify the code such that transactions are removed after they are added to the block.

Theory :

A blockchain is a decentralized digital ledger that records transactions in a secure and immutable manner. In a peer-to-peer (P2P) network, multiple nodes communicate directly with each other without relying on a central authority. Each node maintains its own copy of the blockchain and participates in validating and propagating transactions. However, P2P networks face several challenges such as security threats, double spending, network delays, and scalability issues. Since there is no central controller, the system must ensure that malicious nodes cannot corrupt the network. Blockchain technology addresses these challenges through cryptographic hashing, consensus mechanisms, and distributed validation.

In a blockchain network, transactions represent the transfer of digital currency from one user to another. When a transaction is created, it is broadcast to all nodes in the network. Each node verifies the transaction before accepting it. Valid transactions are temporarily stored in a memory pool (mempool).

Mining in blockchain :

Mining is the process of validating transactions and adding them to the blockchain by creating a new block. It involves solving a computational puzzle known as Proof of Work. The miner must find a valid proof that satisfies specific conditions defined by the hashing algorithm. Once the correct proof is found, a new block is created, linked to the previous block using a cryptographic

hash, and added to the chain. Mining ensures security, prevents tampering, and maintains the integrity of the blockchain. As an incentive, the miner receives a mining reward, which is a predefined amount of cryptocurrency added as a special transaction in the new block.

Steps in Mining

1. Collect transactions from the mempool.
2. Solve a mathematical puzzle.
3. Generate a valid proof.
4. Create a new block.
5. Add the block to the blockchain.

Purpose of Mining

- Validate transactions.
- Secure the network.
- Prevent double spending.
- Maintain decentralization.

Multinode blockchain network :

A multinode blockchain network consists of multiple independent nodes running simultaneously. Each node maintains its own copy of the blockchain and communicates with other nodes through HTTP requests. In this experiment, three nodes were executed on different ports to simulate decentralization. These nodes were connected so that they could share blockchain data and maintain synchronization. This setup demonstrates how decentralized systems operate without a central server. A multinode network contains multiple independent blockchain nodes.

Characteristics

- Each node has:
    - Its own blockchain copy.
    - Its own mempool.
- Nodes communicate using HTTP requests.

In the Experiment

- Three nodes were created:
    - Node 1 (port 5001)
    - Node 2 (port 5002)
    - Node 3 (port 5003)

Advantages

- Decentralization
- Fault tolerance
- Increased security
- No single point of failure

Consensus mechanism :

To ensure that all nodes agree on a single version of the blockchain, a consensus mechanism is used. In this implementation, the Longest Chain Rule is applied. According to this rule, if a node discovers another chain that is longer and valid, it replaces its own chain with the longer one. This process is known as chain replacement. The longest chain is considered the valid chain because it represents the most computational work done through mining. This mechanism resolves conflicts and maintains consistency across the network. Consensus is the process by which nodes agree on the correct blockchain.

Longest Chain Rule

In this experiment:

- Each node checks other nodes' chains.
- The node adopts the longest valid chain.

Why Longest Chain?

- It represents the most computational work.
- Harder for attackers to fake.

Transactions and Mining Reward :

Transactions represent the transfer of cryptocurrency between users. Each transaction contains the sender's address, the receiver's address, and the amount transferred. These transactions are collected by miners and added to new blocks. To encourage miners to participate in the network, a mining reward is provided. When a miner successfully mines a block, a special transaction is added to the block that grants the miner a certain amount of cryptocurrency as a reward.

Chain replacement :

Chain replacement is a process used to maintain consistency across the network. Each node periodically checks the chains of other nodes in the network. If it finds a longer and valid chain, it replaces its own chain with the longer one. This ensures that all nodes eventually agree on the same version of the blockchain. The longest chain is considered the valid one because it represents the most computational work done by the network. Chain replacement ensures all nodes follow the same blockchain.

Process

1. Node checks chains of connected nodes.
2. Finds the longest valid chain.
3. If another node has a longer chain:
    - Replace current chain.
4. If not:
    - Keep current chain.

Purpose

- Resolve conflicts between nodes.
- Maintain a single global blockchain.

Programs And Outputs :

Code :

Hadcoin_node_5001.py

```
# Module 2 - Create a Cryptocurrency
# To be installed:
# Flask==0.12.2: pip install Flask==0.12.2
# Postman HTTP Client: https://www.getpostman.com/
# requests==2.18.4: pip install requests==2.18.4

# Importing the libraries
import datetime
import hashlib
import json
from flask import Flask, jsonify, request
import requests
from uuid import uuid4                                  # Generate a unique id that is in hex
from urllib.parse import urlparse                       # To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []                          # Adding transactions before they are added to a block
        self.create_block(proof = 1, previous_hash = '0')
        self.nodes = set()                              # Set is used as there is no order to be maintained as the
nodes can be from all around the globe

    def create_block(self, proof, previous_hash):
        block = {'index': len(self.chain) + 1,
                 'timestamp': str(datetime.datetime.now()),
                 'proof': proof,
                 'previous_hash': previous_hash,
```

```
                'transactions': self.transactions}                    # Adding transactions to make the blockchain a
cryptocurrency
        self.transactions = []                                        # The list of transacction should become empty after they
are added to a block
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        new_proof = 1
        check_proof = False
        while check_proof is False:
            hash_operation = hashlib.sha256(str(new_proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1
        return new_proof

    def hash(self, block):
        encoded_block = json.dumps(block, sort_keys = True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        previous_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(str(proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] != '0000':
                return False
            previous_block = block
            block_index += 1
        return True
```

```
    # This method will add the trqnsaction to the list of trnsactions
    def add_transaction(self, sender, receiver, amount):
        self.transactions.append({'sender': sender,
                                  'receiver': receiver,
                                  'amount': amount})
        previous_block = self.get_previous_block()
          return previous_block['index'] + 1                                    # It will return the block index to which the
transaction should be added

    # This function will add the node containing an address to the set of nodes created in init function
    def add_node(self, address):
        parsed_url = urlparse(address)                                  # urlparse will parse the url from the address
         self.nodes.add(parsed_url.netloc)                                  # Add is used and not append as it's a set. Netloc will
only return '127.0.0.1:5000'

     # Consensus Protocol. This function will replace all the shorter chain with the longer chain in all the nodes on the
network
    def replace_chain(self):
        network = self.nodes                                            # network variable is the set of nodes all around the globe
          longest_chain = None                                                  # It will hold the longest chain when we scan the
network
          max_length = len(self.chain)                                      # This will hold the length of the chain held by the
node that runs this function
        for node in network:
              response = requests.get(f'http://{node}/get_chain')               # Use get chain method already created to  get
the length of the chain
            if response.status_code == 200:
                  length = response.json()['length']                        # Extract the length of the chain from get_chain
fiunction
              chain = response.json()['chain']
                  if length > max_length and self.is_chain_valid(chain):        # We check if the length is bigger and if the
chain is valid then
                    max_length = length                                     # We update the max length
                    longest_chain = chain                                   # We update the longest chain
        if longest_chain:                                               # If longest_chain is not none that means it was replaced
             self.chain = longest_chain                                     # Replace the chain of the current node with the longest
chain
            return True
        return False                                                    # Return false if current chain is the longest one

# Part 2 - Mining our Blockchain
```

```
# Creating a Web App
app = Flask(__name__)

# Creating an address for the node on Port 5000. We will create some other nodes as well on different ports
node_address = str(uuid4()).replace('-', '')                                    #

# Creating a Blockchain
blockchain = Blockchain()

# Mining a new block
@app.route('/mine_block', methods = ['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']
    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)
      blockchain.add_transaction(sender = node_address, receiver = 'Richard', amount = 1) # Hadcoins to mine the
block (A Reward). So the node gives 1 hadcoin to Abcde for mining the block
    block = blockchain.create_block(proof, previous_hash)
    response = {'message': 'Congratulations, you just mined a block!',
                'index': block['index'],
                'timestamp': block['timestamp'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash'],
                'transactions': block['transactions']}
    return jsonify(response), 200

# Getting the full Blockchain
@app.route('/get_chain', methods = ['GET'])
def get_chain():
    response = {'chain': blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response), 200

# Checking if the Blockchain is valid
@app.route('/is_valid', methods = ['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
```

```
    else:
        response = {'message': 'Houston, we have a problem. The Blockchain is not valid.'}
    return jsonify(response), 200

# Adding a new transaction to the Blockchain
@app.route('/add_transaction', methods = ['POST'])                    # Post method as we have to pass something
to get something in return
def add_transaction():
    json = request.get_json()                                          # This will get the json file from postman. In Postman we
will create a json file in which we will pass the values for the keys in the json file
    transaction_keys = ['sender', 'receiver', 'amount']
    if not all(key in json for key in transaction_keys):               # Checking if all keys are available in json
        return 'Some elements of the transaction are missing', 400
    index = blockchain.add_transaction(json['sender'], json['receiver'], json['amount'])
    response = {'message': f'This transaction will be added to Block {index}'}
    return jsonify(response), 201                                      # Code 201 for creation

# Part 3 - Decentralizing our Blockchain

# Connecting new nodes
@app.route('/connect_node', methods = ['POST'])                        # POST request to register the new nodes
from the json file
def connect_node():
    json = request.get_json()
    nodes = json.get('nodes')                                          # Get the nodes from json file
    if nodes is None:
        return "No node", 400
    for node in nodes:
        blockchain.add_node(node)
     response = {'message': 'All the nodes are now connected. The Hadcoin Blockchain now contains the following
nodes:',
                'total_nodes': list(blockchain.nodes)}
    return jsonify(response), 201

# Replacing the chain by the longest chain if needed
@app.route('/replace_chain', methods = ['GET'])
def replace_chain():
    is_chain_replaced = blockchain.replace_chain()
    if is_chain_replaced:
        response = {'message': 'The nodes had different chains so the chain was replaced by the longest one.',
                    'new_chain': blockchain.chain}
```

```
    else:
        response = {'message': 'All good. The chain is the largest one.',
                    'actual_chain': blockchain.chain}
    return jsonify(response), 200

# Running the app
app.run(host = '0.0.0.0', port = 5001)
```

Hadcoin_node_5002.py

```
# Module 2 - Create a Cryptocurrency
# To be installed:
# Flask==0.12.2: pip install Flask==0.12.2
# Postman HTTP Client: https://www.getpostman.com/
# requests==2.18.4: pip install requests==2.18.4

# Importing the libraries
import datetime
import hashlib
import json
from flask import Flask, jsonify, request
import requests
from uuid import uuid4                                  # Generate a unique id that is in hex
from urllib.parse import urlparse                       # To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []                          # Adding transactions before they are added to a block
        self.create_block(proof = 1, previous_hash = '0')
        self.nodes = set()                              # Set is used as there is no order to be maintained as the
nodes can be from all around the globe

    def create_block(self, proof, previous_hash):
        block = {'index': len(self.chain) + 1,
                 'timestamp': str(datetime.datetime.now()),
                 'proof': proof,
                 'previous_hash': previous_hash,
                     'transactions': self.transactions}          # Adding transactions to make the blockchain a
```

```
cryptocurrency
        self.transactions = []                                  # The list of transacction should become empty after they
are added to a block
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        new_proof = 1
        check_proof = False
        while check_proof is False:
            hash_operation = hashlib.sha256(str(new_proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1
        return new_proof

    def hash(self, block):
        encoded_block = json.dumps(block, sort_keys = True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        previous_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(str(proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] != '0000':
                return False
            previous_block = block
            block_index += 1
        return True

    # This method will add the trqnsaction to the list of trnsactions
```

```
    def add_transaction(self, sender, receiver, amount):
        self.transactions.append({'sender': sender,
                                  'receiver': receiver,
                                  'amount': amount})
        previous_block = self.get_previous_block()
        return previous_block['index'] + 1                                  # It will return the block index to which the transaction should be added

    # This function will add the node containing an address to the set of nodes created in init function
    def add_node(self, address):
        parsed_url = urlparse(address)                                      # urlparse will parse the url from the address
        self.nodes.add(parsed_url.netloc)                                   # Add is used and not append as it's a set. Netloc will only return '127.0.0.1:5000'

    # Consensus Protocol. This function will replace all the shorter chain with the longer chain in all the nodes on the network
    def replace_chain(self):
        network = self.nodes                                                # network variable is the set of nodes all around the globe
        longest_chain = None                                                # It will hold the longest chain when we scan the network
        max_length = len(self.chain)                                        # This will hold the length of the chain held by the node that runs this function
        for node in network:
            response = requests.get(f'http://{node}/get_chain')             # Use get chain method already created to  get the length of the chain
            if response.status_code == 200:
                length = response.json()['length']                          # Extract the length of the chain from get_chain fiunction
                chain = response.json()['chain']
                if length > max_length and self.is_chain_valid(chain):      # We check if the length is bigger and if the chain is valid then
                    max_length = length                                     # We update the max length
                    longest_chain = chain                                   # We update the longest chain
        if longest_chain:                                                   # If longest_chain is not none that means it was replaced
            self.chain = longest_chain                                      # Replace the chain of the current node with the longest chain
            return True
        return False                                                        # Return false if current chain is the longest one

# Part 2 - Mining our Blockchain
```

```
# Creating a Web App
app = Flask(__name__)

# Creating an address for the node on Port 5000. We will create some other nodes as well on different ports
node_address = str(uuid4()).replace('-', '')                                    #

# Creating a Blockchain
blockchain = Blockchain()

# Mining a new block
@app.route('/mine_block', methods = ['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']
    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)
    blockchain.add_transaction(sender = node_address, receiver = 'you', amount = 1) # Hadcoins to mine the block (A
Reward). So the node gives 1 hadcoin to Abcde for mining the block
    block = blockchain.create_block(proof, previous_hash)
    response = {'message': 'Congratulations, you just mined a block!',
                'index': block['index'],
                'timestamp': block['timestamp'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash'],
                'transactions': block['transactions']}
    return jsonify(response), 200

# Getting the full Blockchain
@app.route('/get_chain', methods = ['GET'])
def get_chain():
    response = {'chain': blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response), 200

# Checking if the Blockchain is valid
@app.route('/is_valid', methods = ['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
```

```
        response = {'message': 'Houston, we have a problem. The Blockchain is not valid.'}
    return jsonify(response), 200

# Adding a new transaction to the Blockchain
@app.route('/add_transaction', methods = ['POST'])                    # Post method as we have to pass something
to get something in return
def add_transaction():
    json = request.get_json()                                          # This will get the json file from postman. In Postman we
will create a json file in which we will pass the values for the keys in the json file
    transaction_keys = ['sender', 'receiver', 'amount']
    if not all(key in json for key in transaction_keys):               # Checking if all keys are available in json
        return 'Some elements of the transaction are missing', 400
    index = blockchain.add_transaction(json['sender'], json['receiver'], json['amount'])
    response = {'message': f'This transaction will be added to Block {index}'}
    return jsonify(response), 201                                      # Code 201 for creation

# Part 3 - Decentralizing our Blockchain

# Connecting new nodes
@app.route('/connect_node', methods = ['POST'])                        # POST request to register the new nodes
from the json file
def connect_node():
    json = request.get_json()
    nodes = json.get('nodes')                                          # Get the nodes from json file
    if nodes is None:
        return "No node", 400
    for node in nodes:
        blockchain.add_node(node)
     response = {'message': 'All the nodes are now connected. The Hadcoin Blockchain now contains the following
nodes:',
                'total_nodes': list(blockchain.nodes)}
    return jsonify(response), 201

# Replacing the chain by the longest chain if needed
@app.route('/replace_chain', methods = ['GET'])
def replace_chain():
    is_chain_replaced = blockchain.replace_chain()
    if is_chain_replaced:
        response = {'message': 'The nodes had different chains so the chain was replaced by the longest one.',
                    'new_chain': blockchain.chain}
    else:
```

```
        response = {'message': 'All good. The chain is the largest one.',
                    'actual_chain': blockchain.chain}
    return jsonify(response), 200

# Running the app
app.run(host = '0.0.0.0', port = 5002)
```

Hadcoin_node_5003.py

```
# Module 2 - Create a Cryptocurrency
# To be installed:
# Flask==0.12.2: pip install Flask==0.12.2
# Postman HTTP Client: https://www.getpostman.com/
# requests==2.18.4: pip install requests==2.18.4

# Importing the libraries
import datetime
import hashlib
import json
from flask import Flask, jsonify, request
import requests
from uuid import uuid4                                  # Generate a unique id that is in hex
from urllib.parse import urlparse                       # To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []                          # Adding transactions before they are added to a block
        self.create_block(proof = 1, previous_hash = '0')
        self.nodes = set()                              # Set is used as there is no order to be maintained as the
nodes can be from all around the globe

    def create_block(self, proof, previous_hash):
        block = {'index': len(self.chain) + 1,
                 'timestamp': str(datetime.datetime.now()),
                 'proof': proof,
                 'previous_hash': previous_hash,
                     'transactions': self.transactions}          # Adding transactions to make the blockchain a
cryptocurrency
```

```
        self.transactions = []                                  # The list of transacction should become empty after they
are added to a block
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        new_proof = 1
        check_proof = False
        while check_proof is False:
            hash_operation = hashlib.sha256(str(new_proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1
        return new_proof

    def hash(self, block):
        encoded_block = json.dumps(block, sort_keys = True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        previous_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(str(proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] != '0000':
                return False
            previous_block = block
            block_index += 1
        return True

    # This method will add the trqnsaction to the list of trnsactions
    def add_transaction(self, sender, receiver, amount):
```

```
        self.transactions.append({'sender': sender,
                                  'receiver': receiver,
                                  'amount': amount})
        previous_block = self.get_previous_block()
        return previous_block['index'] + 1                    # It will return the block index to which the
transaction should be added

    # This function will add the node containing an address to the set of nodes created in init function
    def add_node(self, address):
        parsed_url = urlparse(address)                        # urlparse will parse the url from the address
        self.nodes.add(parsed_url.netloc)                     # Add is used and not append as it's a set. Netloc will
only return '127.0.0.1:5000'

    # Consensus Protocol. This function will replace all the shorter chain with the longer chain in all the nodes on the
network
    def replace_chain(self):
        network = self.nodes                                  # network variable is the set of nodes all around the globe
        longest_chain = None                                  # It will hold the longest chain when we scan the
network
        max_length = len(self.chain)                          # This will hold the length of the chain held by the
node that runs this function
        for node in network:
            response = requests.get(f'http://{node}/get_chain')          # Use get chain method already created to  get
the length of the chain
            if response.status_code == 200:
                length = response.json()['length']                       # Extract the length of the chain from get_chain
fiunction
                chain = response.json()['chain']
                if length > max_length and self.is_chain_valid(chain):   # We check if the length is bigger and if the
chain is valid then
                    max_length = length                       # We update the max length
                    longest_chain = chain                     # We update the longest chain
        if longest_chain:                                     # If longest_chain is not none that means it was replaced
            self.chain = longest_chain                        # Replace the chain of the current node with the longest
chain
            return True
        return False                                          # Return false if current chain is the longest one

# Part 2 - Mining our Blockchain

# Creating a Web App
```

```
app = Flask(__name__)

# Creating an address for the node on Port 5000. We will create some other nodes as well on different ports
node_address = str(uuid4()).replace('-', '')                                    #

# Creating a Blockchain
blockchain = Blockchain()

# Mining a new block
@app.route('/mine_block', methods = ['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']
    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)
     blockchain.add_transaction(sender = node_address, receiver = 'Abcde', amount = 1) # Hadcoins to mine the block
(A Reward). So the node gives 1 hadcoin to Abcde for mining the block
    block = blockchain.create_block(proof, previous_hash)
    response = {'message': 'Congratulations, you just mined a block!',
                'index': block['index'],
                'timestamp': block['timestamp'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash'],
                'transactions': block['transactions']}
    return jsonify(response), 200

# Getting the full Blockchain
@app.route('/get_chain', methods = ['GET'])
def get_chain():
    response = {'chain': blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response), 200

# Checking if the Blockchain is valid
@app.route('/is_valid', methods = ['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Houston, we have a problem. The Blockchain is not valid.'}
```

```
    return jsonify(response), 200

# Adding a new transaction to the Blockchain
@app.route('/add_transaction', methods = ['POST'])                    # Post method as we have to pass something
to get something in return
def add_transaction():
    json = request.get_json()                                         # This will get the json file from postman. In Postman we
will create a json file in which we will pass the values for the keys in the json file
    transaction_keys = ['sender', 'receiver', 'amount']
    if not all(key in json for key in transaction_keys):              # Checking if all keys are available in json
        return 'Some elements of the transaction are missing', 400
    index = blockchain.add_transaction(json['sender'], json['receiver'], json['amount'])
    response = {'message': f'This transaction will be added to Block {index}'}
    return jsonify(response), 201                                     # Code 201 for creation

# Part 3 - Decentralizing our Blockchain

# Connecting new nodes
@app.route('/connect_node', methods = ['POST'])                       # POST request to register the new nodes
from the json file
def connect_node():
    json = request.get_json()
    nodes = json.get('nodes')                                         # Get the nodes from json file
    if nodes is None:
        return "No node", 400
    for node in nodes:
        blockchain.add_node(node)
     response = {'message': 'All the nodes are now connected. The Hadcoin Blockchain now contains the following
nodes:',
                'total_nodes': list(blockchain.nodes)}
    return jsonify(response), 201

# Replacing the chain by the longest chain if needed
@app.route('/replace_chain', methods = ['GET'])
def replace_chain():
    is_chain_replaced = blockchain.replace_chain()
    if is_chain_replaced:
        response = {'message': 'The nodes had different chains so the chain was replaced by the longest one.',
                    'new_chain': blockchain.chain}
    else:
        response = {'message': 'All good. The chain is the largest one.',
```

```
                'actual_chain': blockchain.chain}
    return jsonify(response), 200

# Running the app
app.run(host = '0.0.0.0', port = 5003)
```

Nodes.json

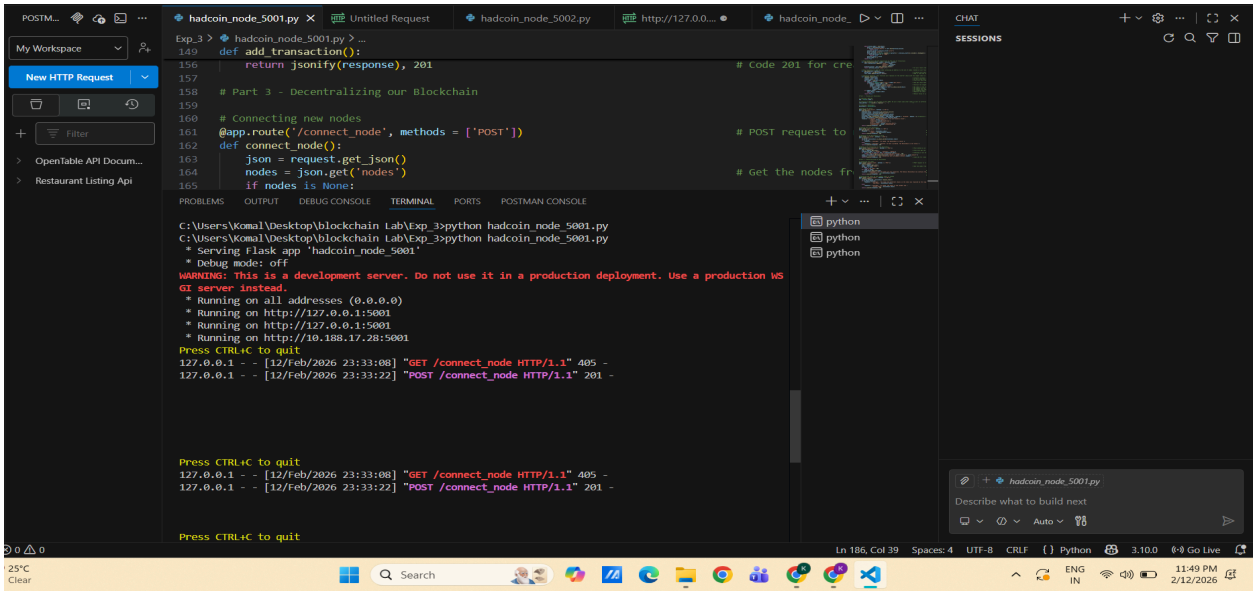
```
{
    "nodes": ["http://127.0.0.1:5001",
              "http://127.0.0.1:5002",
              "http://127.0.0.1:5003"]
}
```

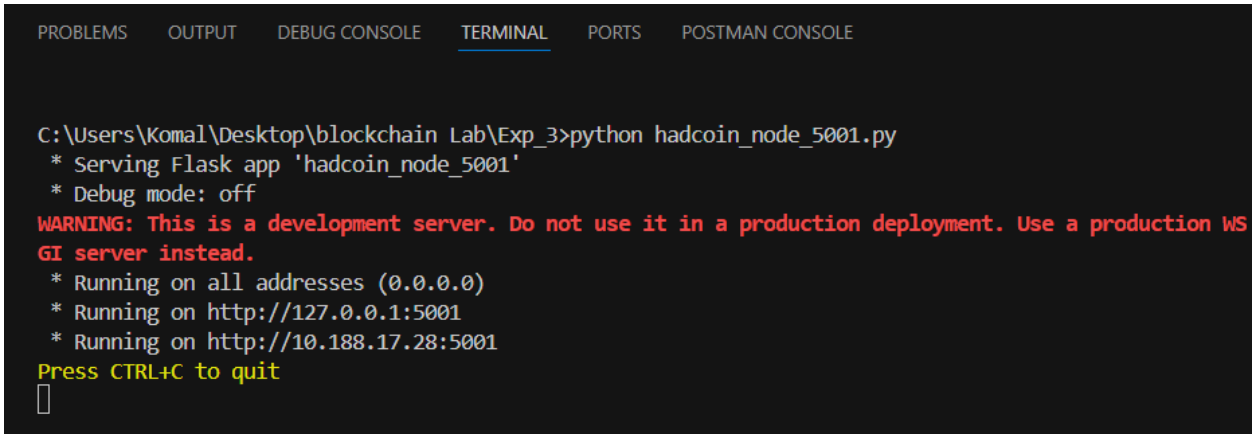
Transaction.json

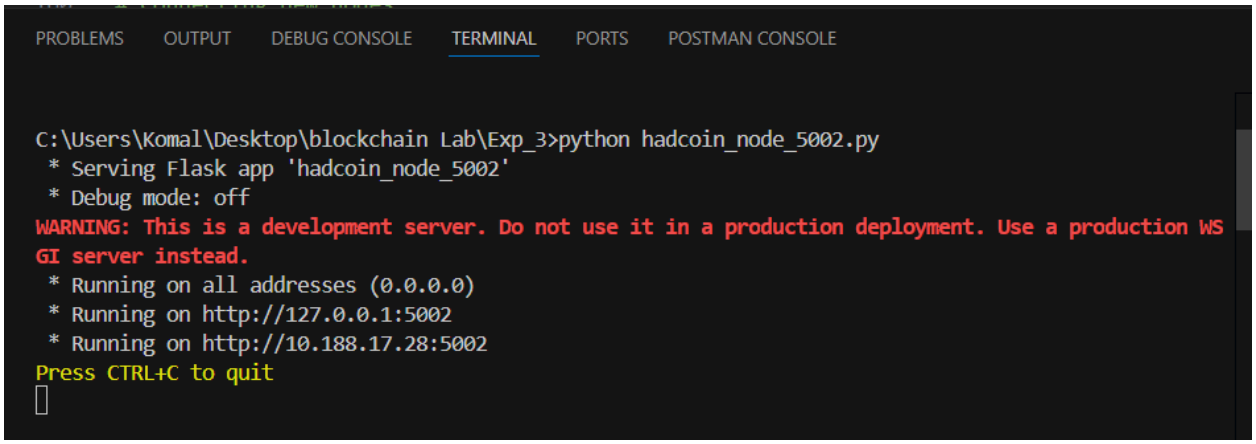
```
{
    "sender": "",
    "receiver": "",
    "amount":
}
```

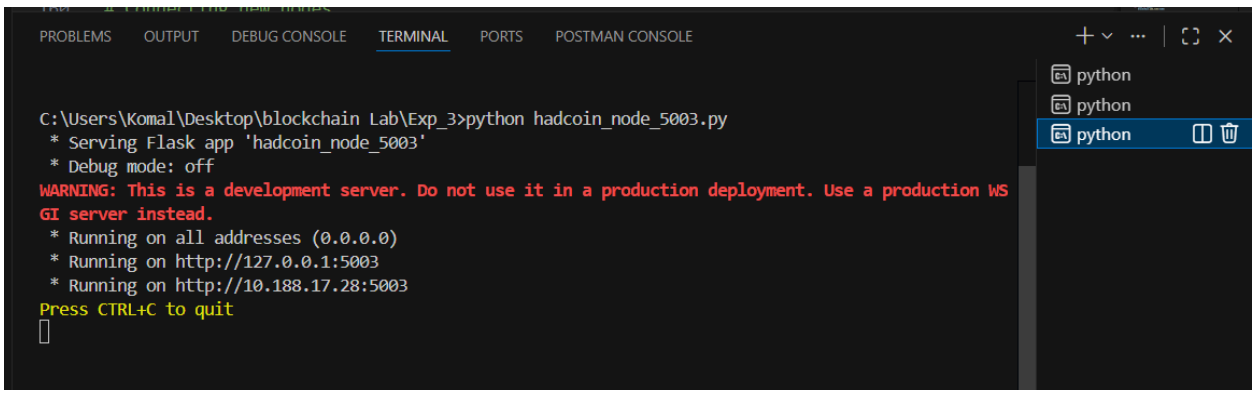
Output :

After installing all the required libraries run three of the files from three different terminals. It will act as three different nodes.



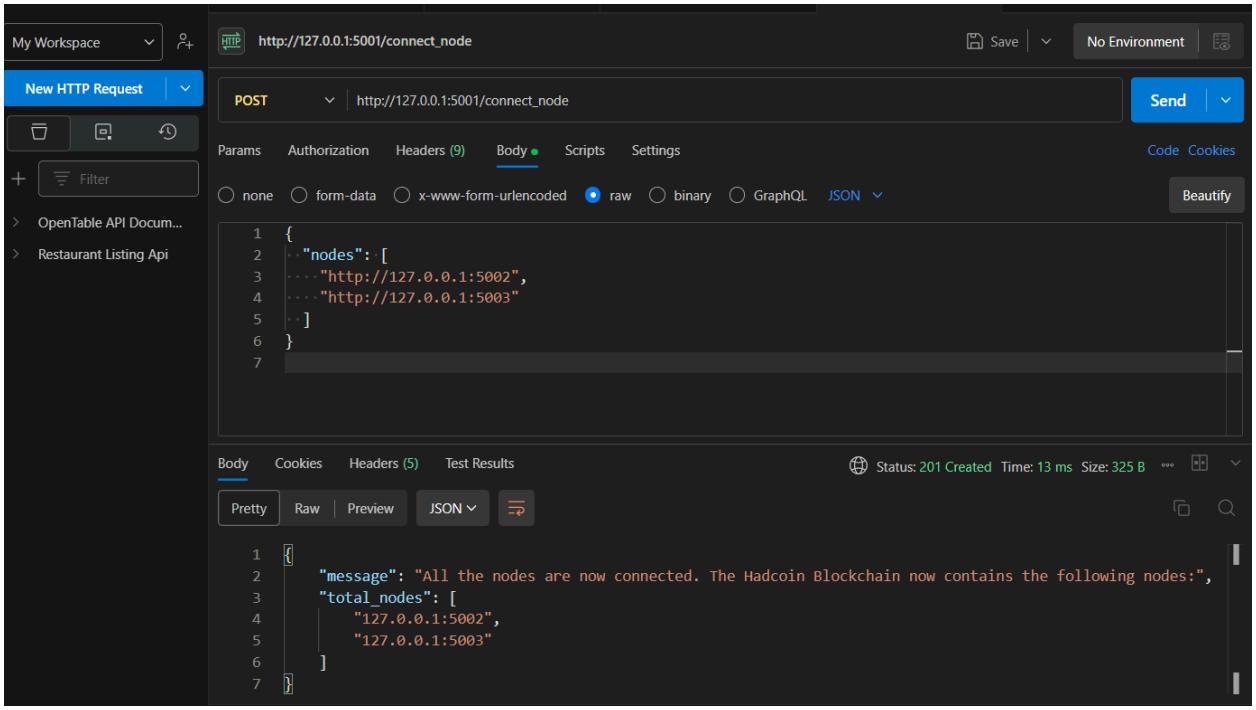


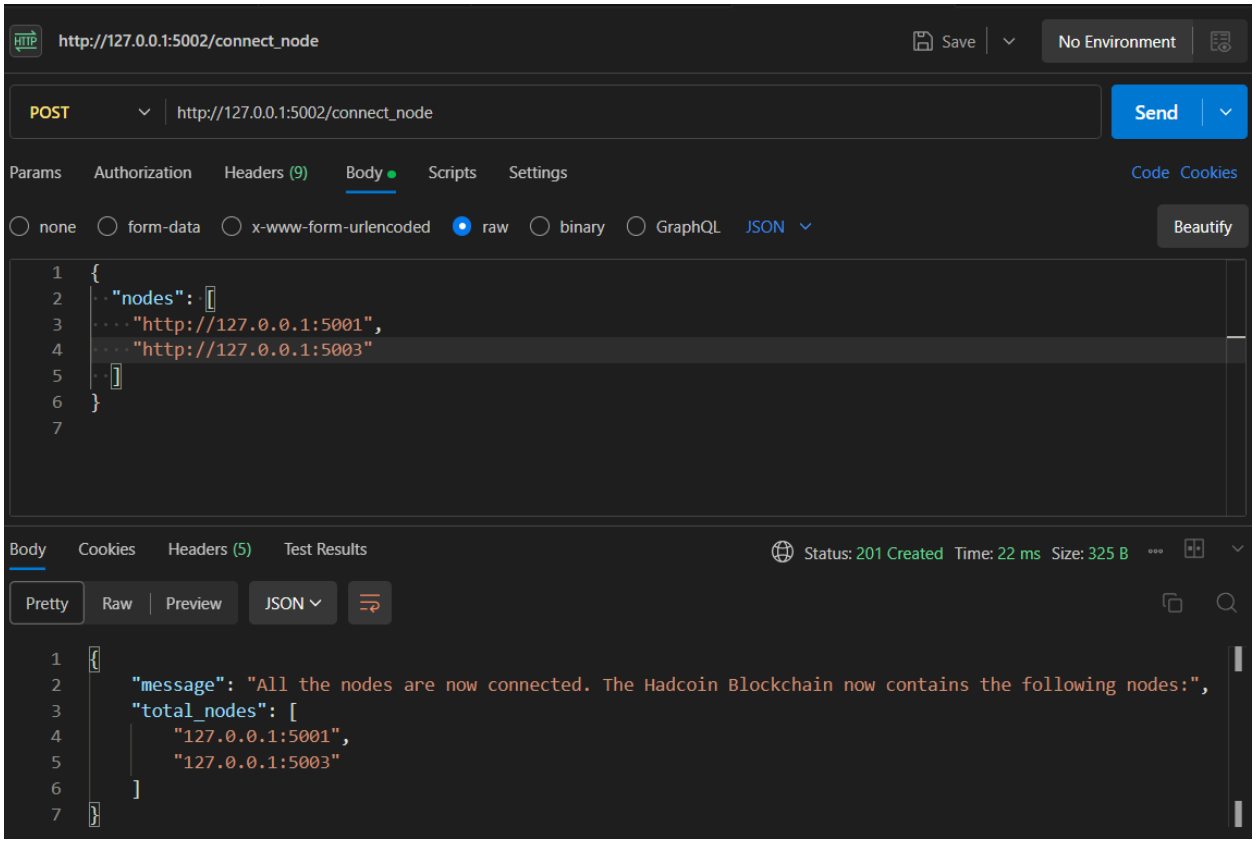


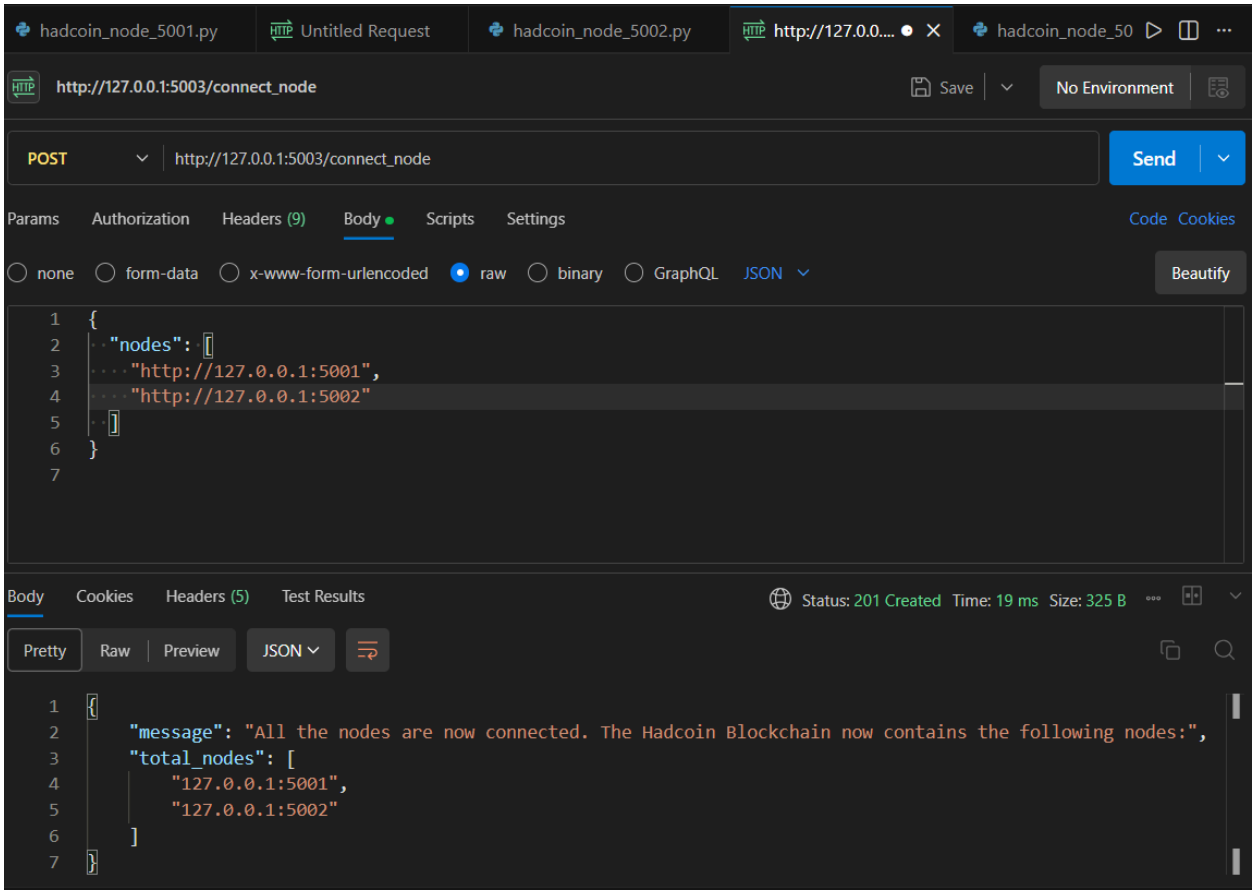


Open Postman and then

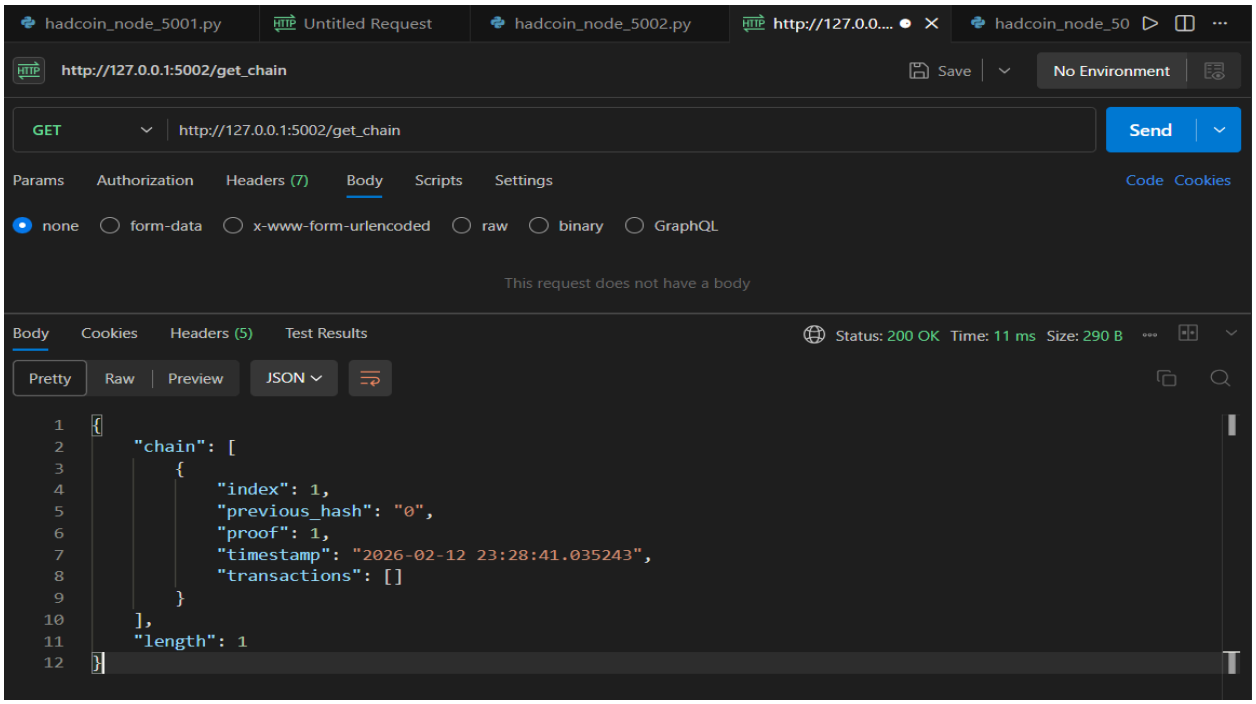
Connect the Nodes (Each node must know about the others) by doing POST request on connect_node by providing other two end point as body

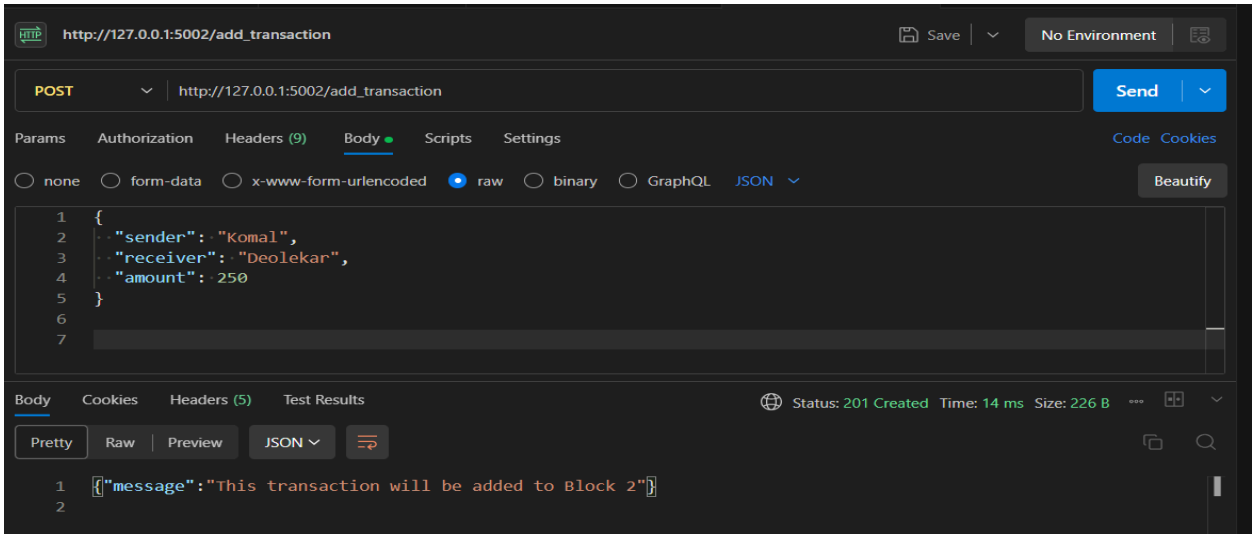

My Workspace
New HTTP Request
Filter
OpenTable API Docum...
Restaurant Listing Api
http://127.0.0.1:5001/connect_node
Save
No Environment
POST
http://127.0.0.1:5001/connect_node
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Code
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nodes": [
"http://127.0.0.1:5002",
"http://127.0.0.1:5003"
]
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 13 ms
Size: 325 B
Pretty
Raw
Preview
JSON
{
"message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",
"total_nodes": [
"127.0.0.1:5002",
"127.0.0.1:5003"
]
}

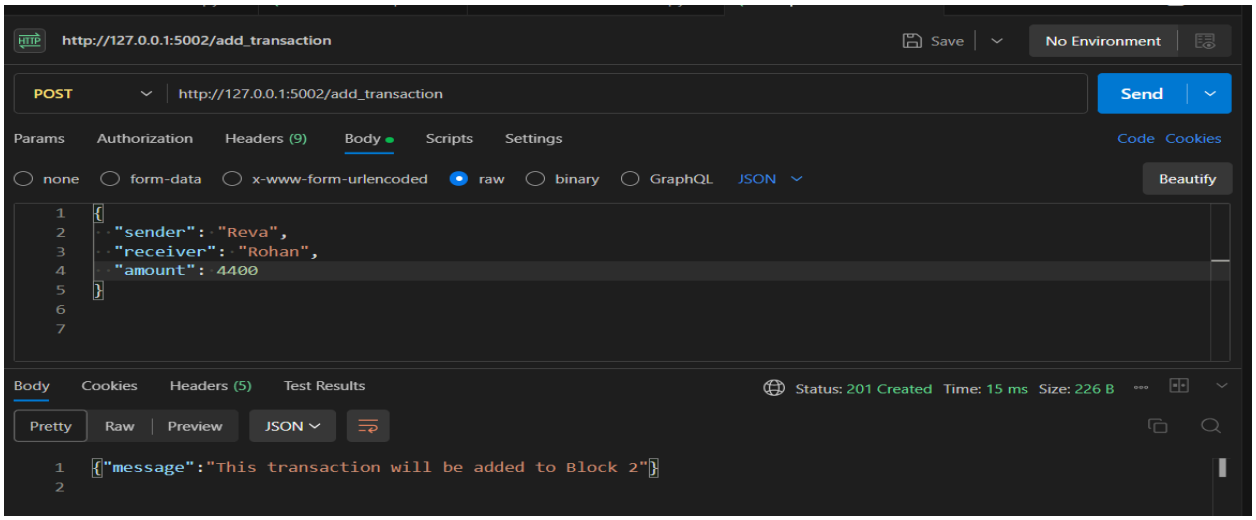

http://127.0.0.1:5002/connect_node
Save
No Environment
POST
http://127.0.0.1:5002/connect_node
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Code
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nodes": [
"http://127.0.0.1:5001",
"http://127.0.0.1:5003"
]
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 22 ms
Size: 325 B
Pretty
Raw
Preview
JSON
{
"message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",
"total_nodes": [
"127.0.0.1:5001",
"127.0.0.1:5003"
]
}



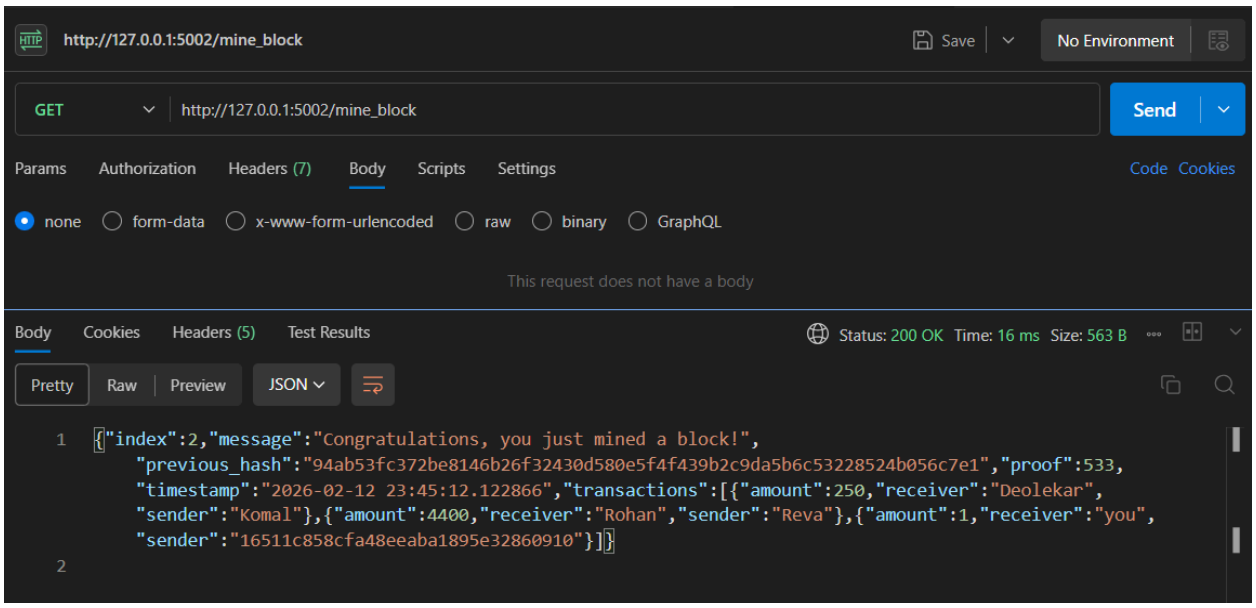
Add transactions from one of the node (I added in node 2)



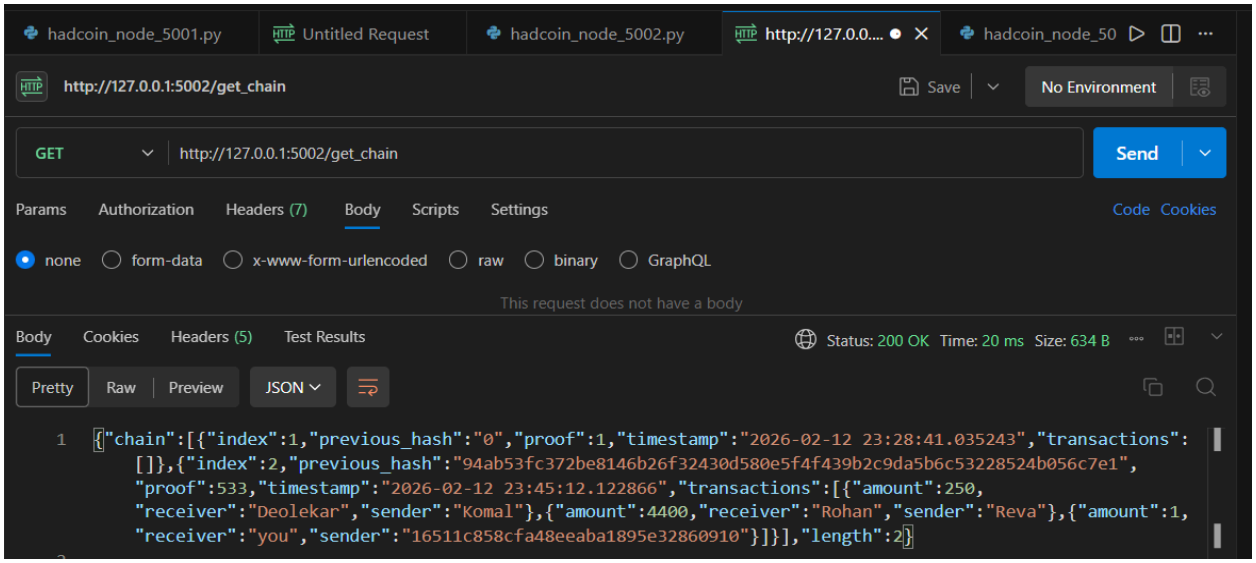




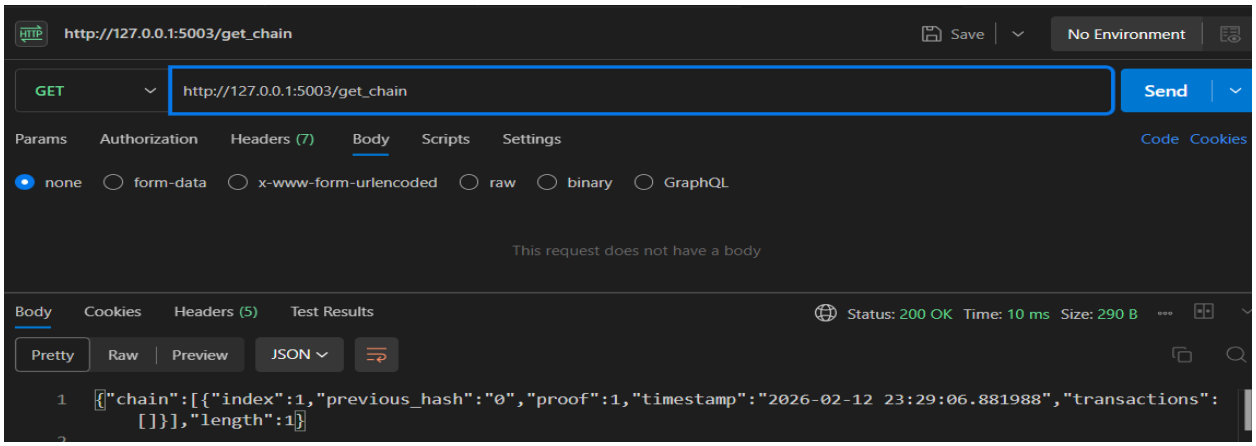
Mine the block



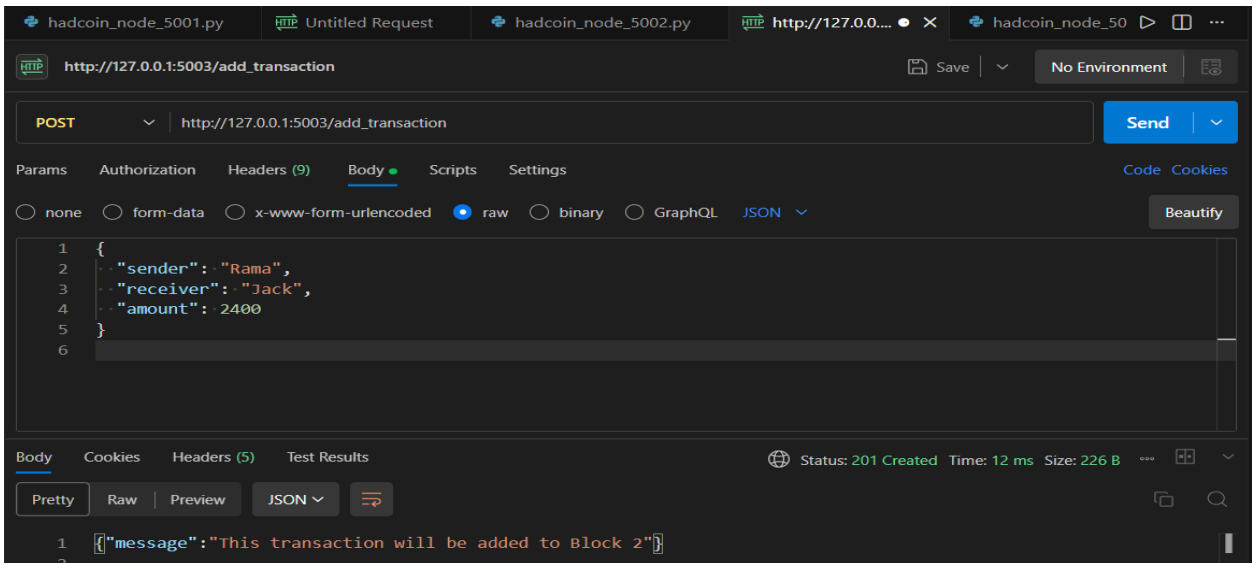
From node 2 I added only one block so length on node 2 id 2(genesis + this block)

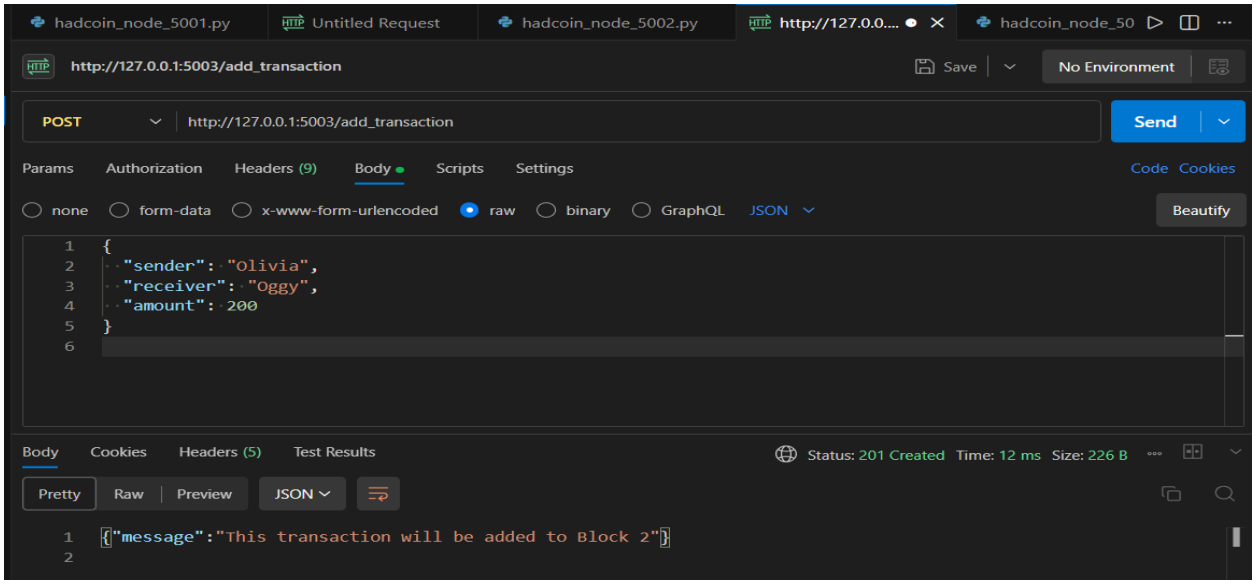


From node 3 I added 4 more blocks so that length will be 5

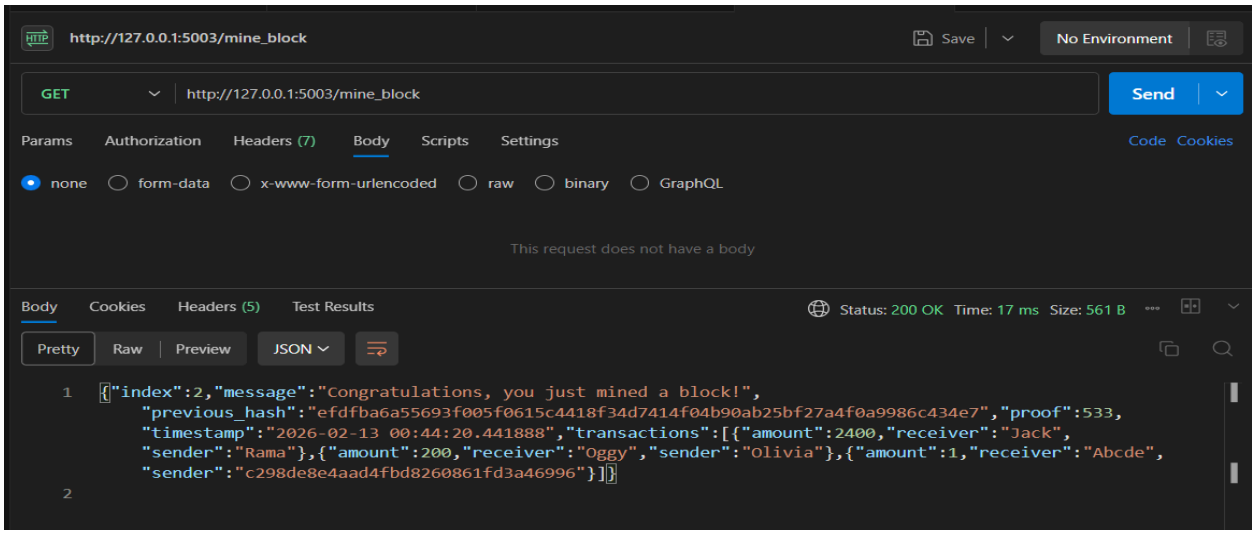


Add transactions

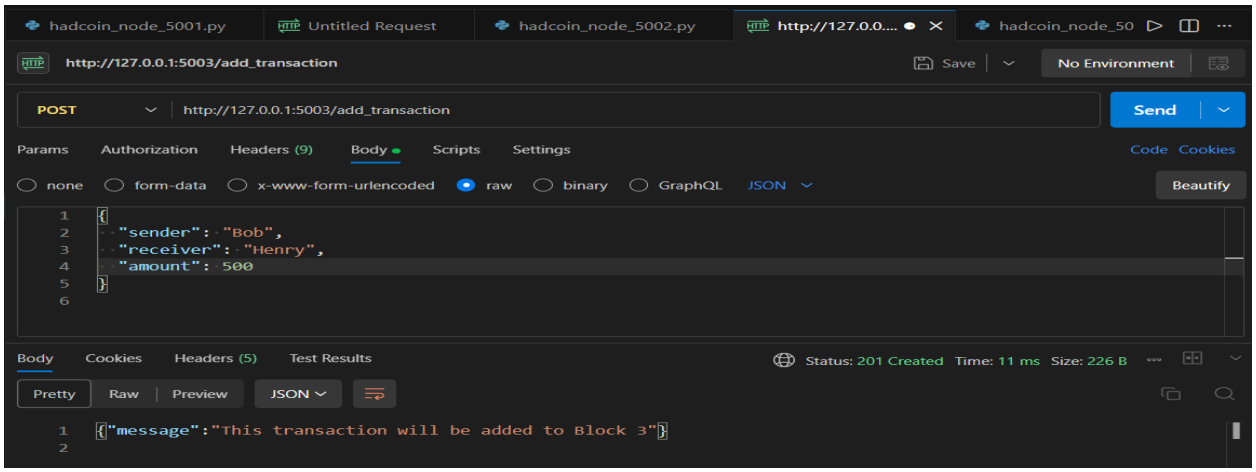




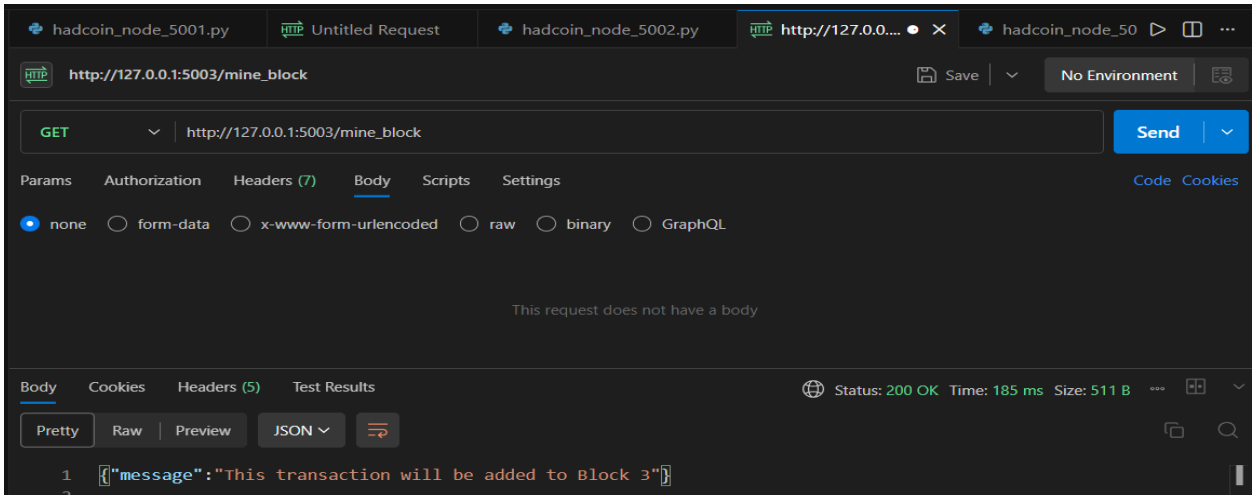
Mine the block



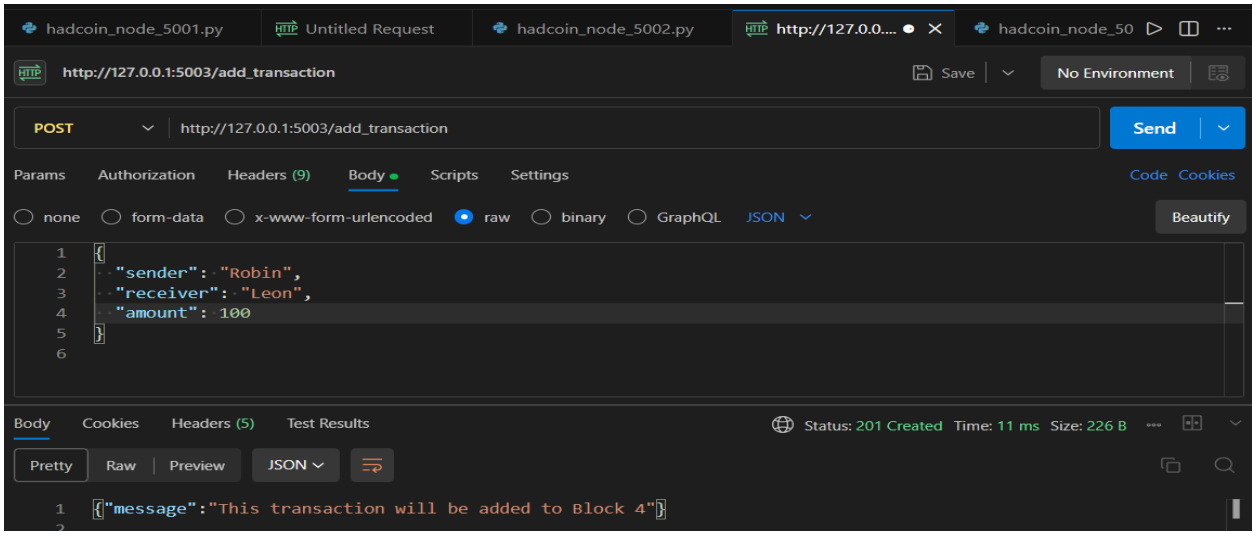
Add transaction



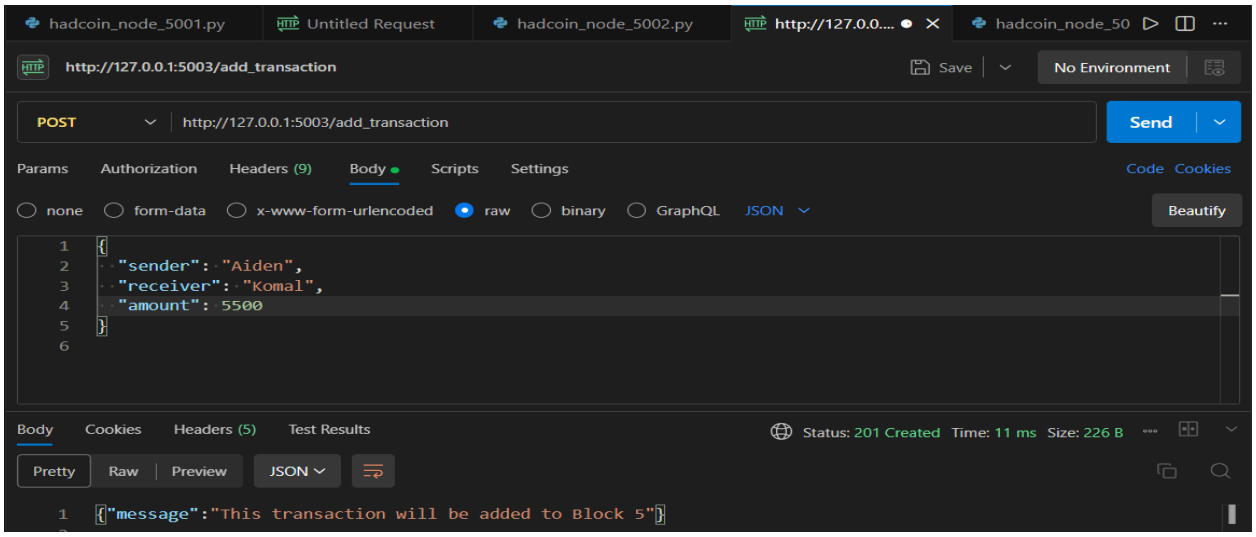
Mine the block



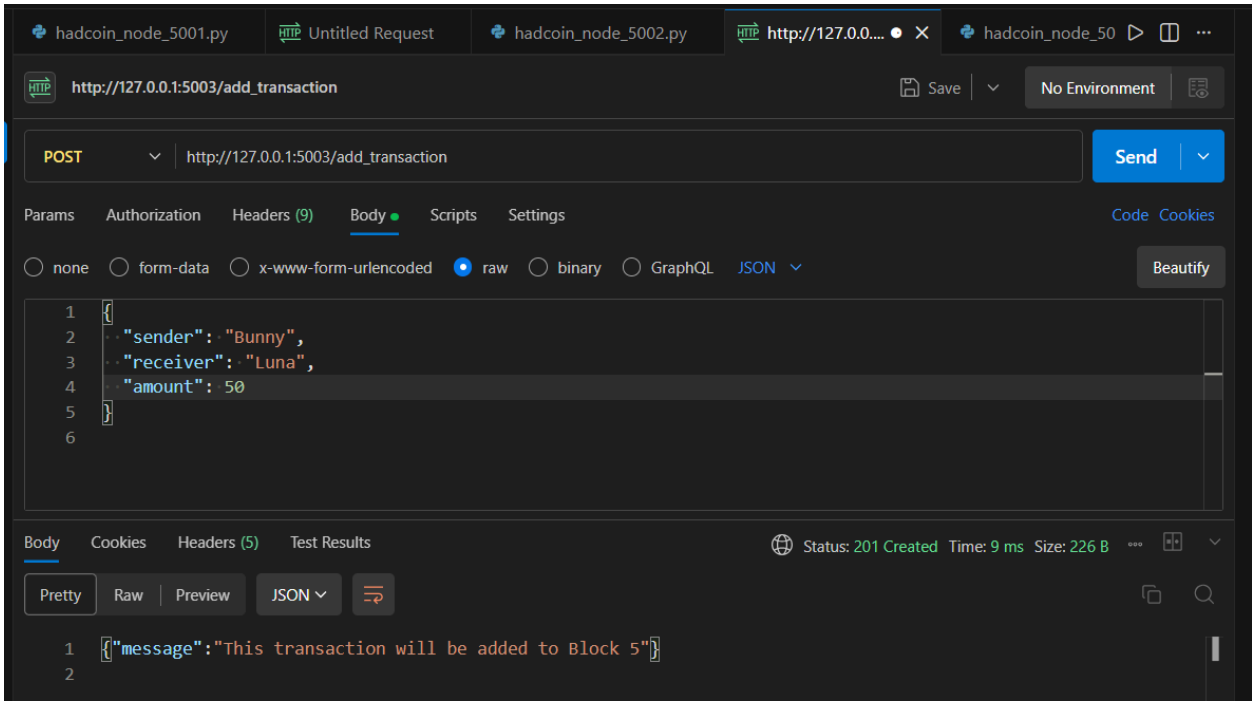
Add transaction



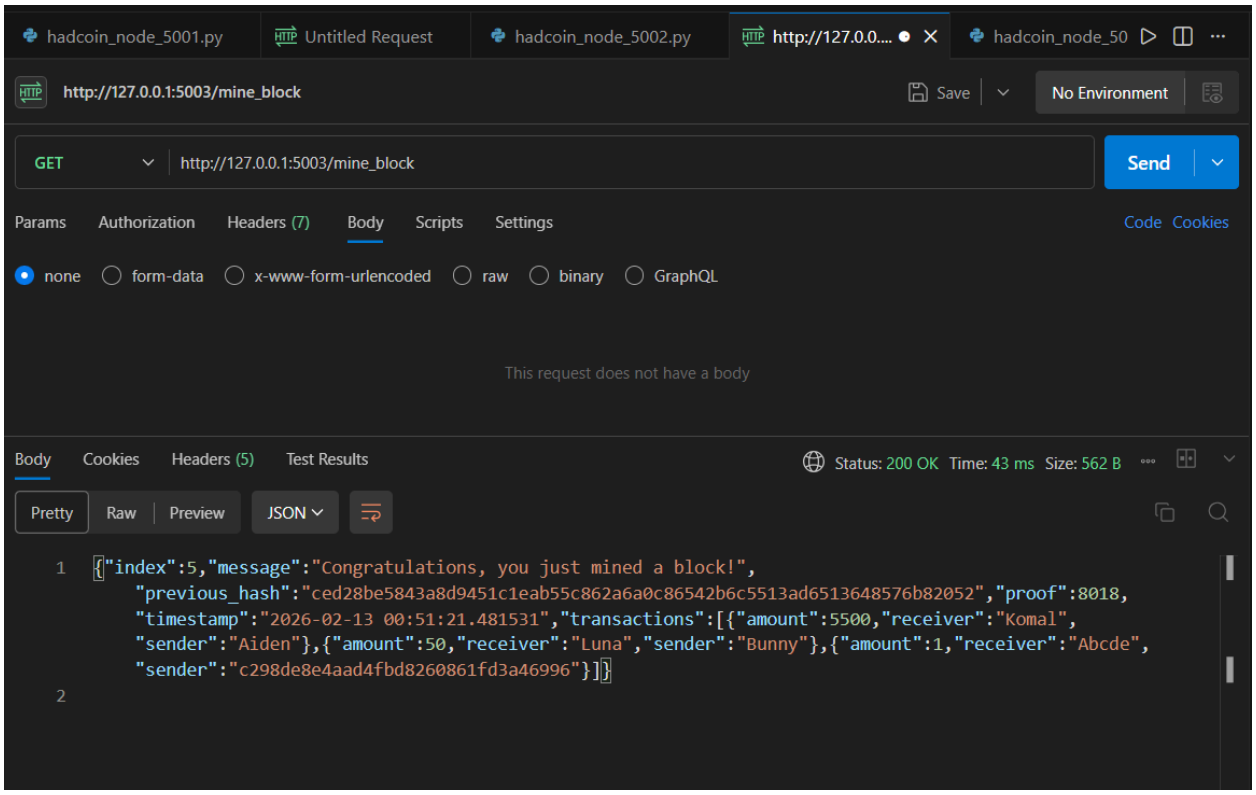
Mine the block



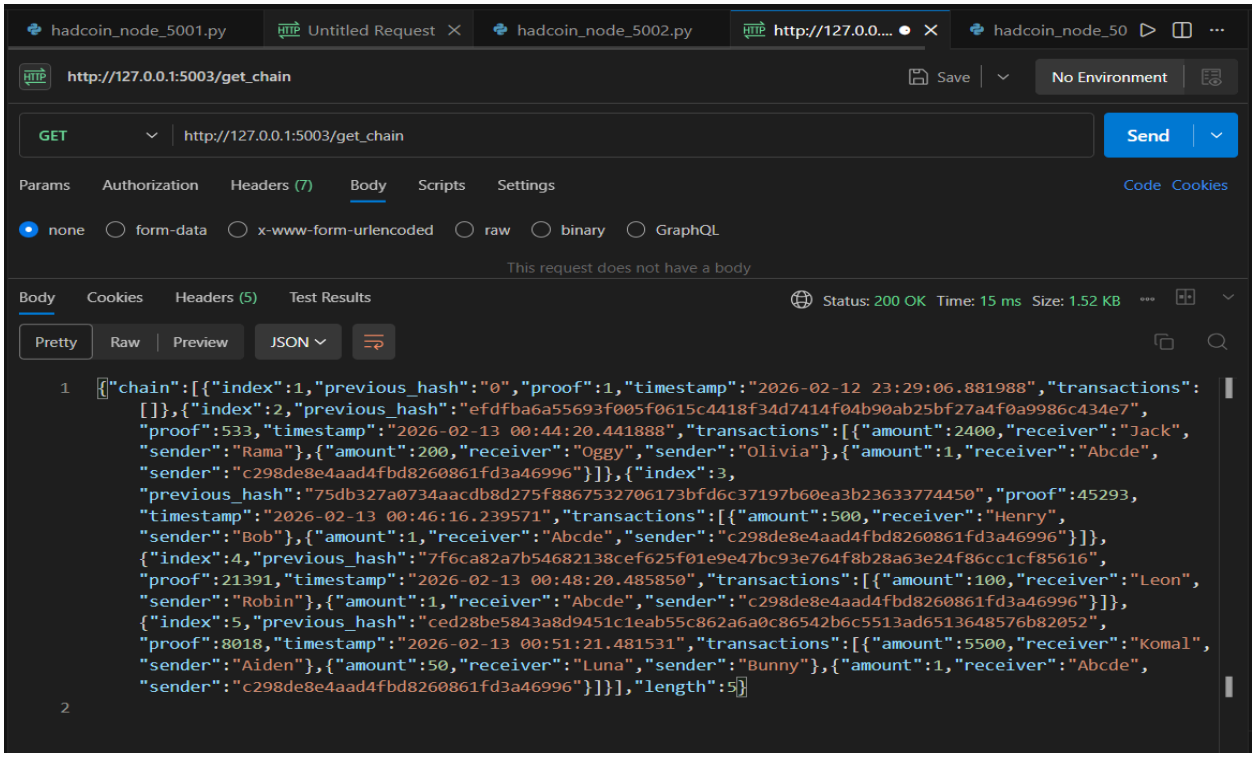
Add transaction



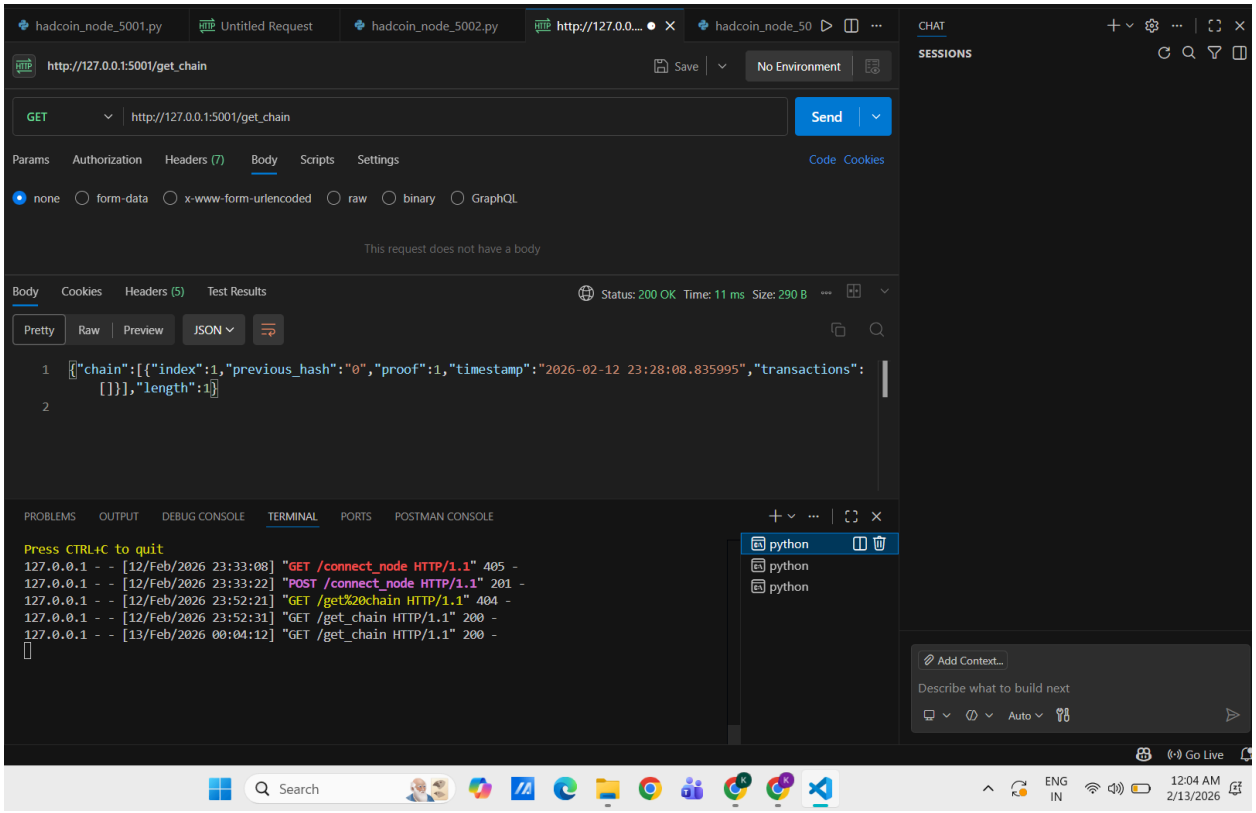
Mine the block



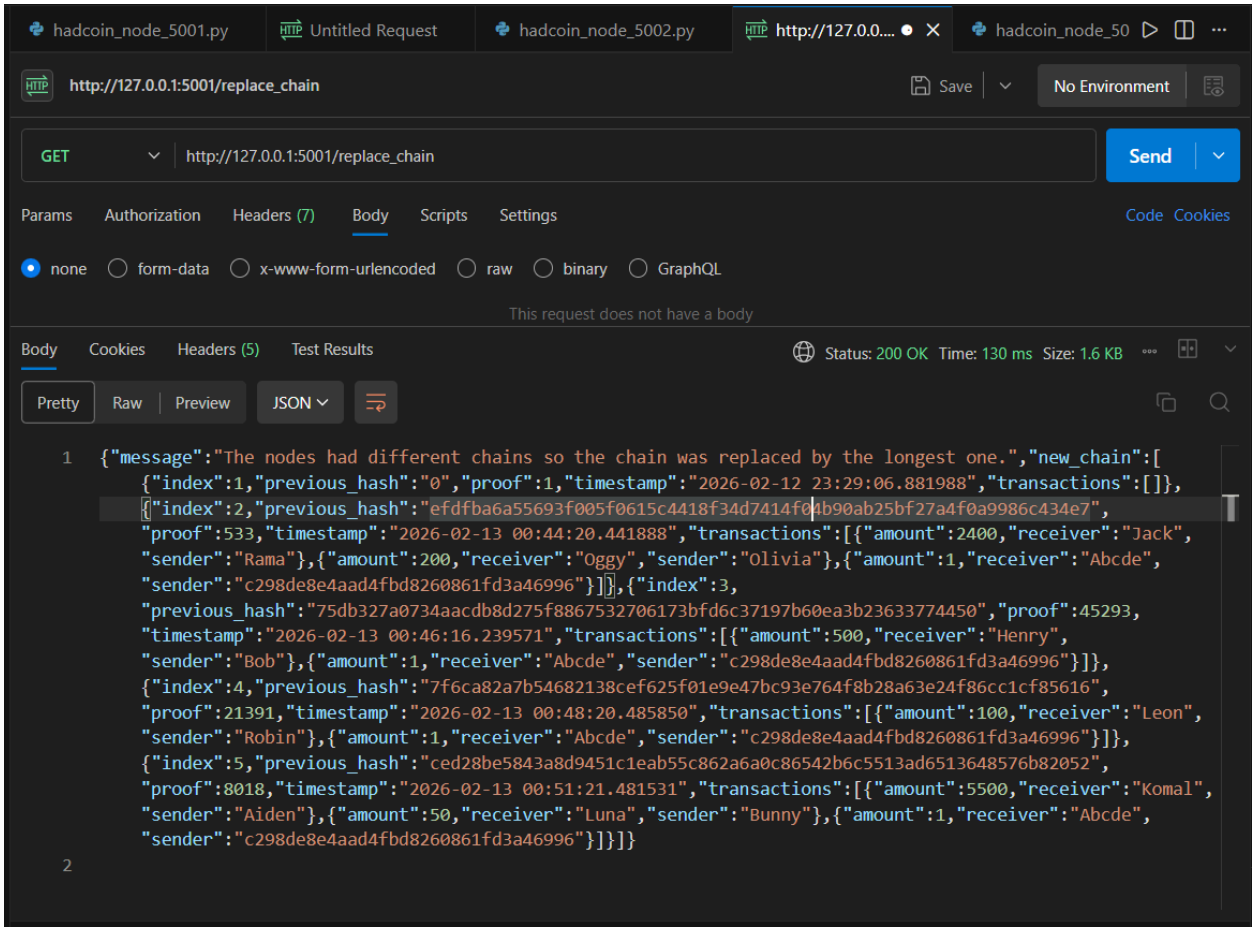
Now we have total 5 blocks on node 3 which is greater than node 2



From node 1 I did not add anything



replace_chain : As node 2 have two blocks and node 3 has five blocks so node replaced its chain with 5 blocks as it is the longest one



Conclusion :

In this experiment, a basic cryptocurrency was created using Python and the Flask framework. A blockchain was implemented with features like transactions, mining using proof of work, and chain validation. Multiple nodes were connected to simulate a decentralized network, and the longest chain consensus mechanism was applied. Transactions were also cleared after being added to a block, preventing duplication. This experiment helped in understanding the core concepts of blockchain, mining, and decentralized systems.